19940020900

N94-25382

SOFTWARE ENGINEERING METHODOLOGIES AND TOOLS

Final Report
NASA/ASEE Summer Faculty Fellowship Program--1993
Johnson Space Center

Prepared By:	Lawrance (Larry) M. Wilcox
Academic Rank	Assistant Professor
University & Department	Southeastern Oklahoma State University (SOSU) Computer Science/Information Systems Department Durant, OK 74701

NASA/JSC

Directorate:	Engineering
Division:	Tracking and Communications
Branch:	Systems Engineering
JSC Colleague:	Sally Stokes, EE
Date Submitted:	July 30, 1993
Contract Number:	NGT-44-001-800

ABSTRACT

The Engineering Profession is reported to be the second oldest profession of humankind. The first engineering discipline was mechanical engineering to support the building of war machines. Over the years many engineering disciplines have developed including chemical, electronic, etc. Common to all engineering disciplines is the use of rigor, models, metrics and predefined methodologies.

Recently a new engineering discipline has appeared on the scene, called Software Engineering. For over thirty years computer software has been developed and the track record has not been good. Software development projects often miss schedules, often are over budget, often does not give the user what is wanted and it has defects. One estimate is there are 1 to 3 defects per 1000 lines of deployed code. More and more systems are requiring larger and more complex software for support. As this requirement grows the software development problems grow exponentially. It is believed that software quality can be improved by applying engineering principles.

Another compelling reason to bring the engineering disciplines to software development is productivity. It has been estimated that productivity of producing software has only increased 1 to 2% a year in the last thirty years. Ironically, the computer and its software have contributed significantly to the industry-wide productivity. But, as the old saw goes "The carpenter's house needs repairing and the cobbler's children need shoes," computer professionals have done a poor job of using the computer to do their job.

Engineering disciplines and methodologies are now emerging supported by software tools that address the problems of software development. This paper addresses some of the current software engineering methodologies as a backdrop for the general evaluation of Computer Assisted Software Engineering (CASE) tools from actual installation of and experimentation with some specific tools.

INTRODUCTION

The Engineering Profession is reported to be the second oldest profession of humankind. The first engineering discipline was mechanical engineering to support the building of war machines. Over the years many different engineering disciplines have developed including chemical, electronic, mechanical, civil, etc. to name a few. Common to all engineering disciplines is the use of rigor, modeling, metrics, tools and predefined methodologies. Recently a new engineering discipline has appeared on the scene, called **Software Engineering.** For over thirty years computer software has been being developed and the track record has not been good. Software development projects often miss schedules, are over budget, do not give the user what they want and have defects. One estimate is there are 1 to 3 defects per 1000 lines of deployed code.[1] More and more systems are requiring larger and more complex software for support. As this requirement grows the software development problems grow exponentially. It is believed that software quality can be improved by applying engineering principles. Another compelling reason to bring the engineering discipline to software development is productivity. It is estimated that productivity of producing software has only increased 1 to 2% a year. The average software developer produces 10 to 15 statements a day. This is to be contrasted with the 1960's estimate of 6 to 10 statements a day. It is interesting to note that in the survey work for these statistics, the programming language being used didn't seem to be a factor. This observation has been used by pundits to advocate using the highest level programming language possible.

Software engineering has now emerged as a discipline in its own right. Although its principal base is in computer science, it also makes use of mathematics, psychology, ergonomics, and management science. The software engineer must be able to assess and apply existing computing techniques in a cost-effective and stable way. She or he applies existing knowledge, derived from more fundamental subjects, in the same way as the electrical or mechanical engineer applies physics and mathematics.

This paper addresses some of the current software engineering methodologies as a backdrop to set the stage for the general evaluation of Computer-Aided Software Engineering (CASE) tools and report on the actual installation and experimentation with some specific software engineering tools. For the purposes of this paper, Software Engineering is defined as a strategy for productively producing quality software.

Software Quality

What is quality software? Characteristics of software quality depends on who is viewing the software. Users judge software to be of high quality if it does what they want in a way that is easy to learn and easy to use. Those who design, write and maintain the software also judge the software quality. Software quality characteristics to these include the correct and efficient use of computer resources, and the relative ease with which the system can be designed, coded, tested and maintained. A third view of software quality is from those who must pay for the software systems, i.e. the cost of producing it.

Software Development Processes

Over the years those who produce software systems have developed techniques that follow the engineering approach of decomposing large complex tasks into smaller less complex tasks, often referred to as the Software Development Process or System Development Life Cycle (SDLC), or Software Development Methodology. These methodologies can be represented by several models.

One of the most popular is the Waterfall model. Although there are no agreed to standards for this model, they all contain at least the following activities:

1. Requirements analysis
2. System Design
3. Program Design
4. Program implementation
5. Unit testing
6. Integration testing
7. System testing
8. System delivery
9. Maintenance

Different software development organizations have adopted various combinations of the above processes. Small organizations tend to be relatively informal and the project proceeds from requirements analysis through design and implementation without much fuss. In the larger organizations, however, things are done on a more formal basis. The above activities are defined as project phases with activities and deliverables defined for each phase. Each project must formally follow a rigid set of procedures to proceed from one phase to another. At each phase exit "management" makes one of three decisions.

1. The project is allowed to proceed to the next phase.
2. The project is killed.
3. The project team is sent back to the drawing board (probably the worst case from a project management point of view).

It is within these Software Development Processes that certain methodologies have evolved over time. This evolution and projected future methodologies is illustrated in the following table:

Table 1. SOFTWARE ENGINEERING METHODOLOGIES

MAJOR TASKS	1950 - 1960's	1970 - 1980's	1980 - 1990's	1990 - 2000's
Analysis	None	Victorian novel*	Structured	Object Oriented Analysis (OOA)
Design	None	Structured	Structured	Object Oriented Design (OOD)
Programming	None	Structured	Structured/ Object Oriented Programming (OOP)	OOP/CASE Generated

* Victorian novel means an extremely large narration of the requirements in English in which you had to read the entire document to get an understanding of the analyst's understanding of the user's requirements. Like a Victorian novel, if you didn't read the last page, you had no idea how the story ended.

In structured oriented analysis and design methodologies, primary emphasis is placed on specifying and decomposing system functionality. The object oriented analysis and design approach focuses first on identifying objects from the application domain, then fitting procedures around them. Just 5 to 10% of commercial application developers use structured analysis and design methodologies, according to CASE expert Yourdon. Most all use structured programming. A few leading edge organizations are moving to object oriented analysis, design, and programming. Indeed, many CASE vendors are joining the object oriented hype.

As the graphical modeling techniques of structured analysis began to spread through systems development organizations in the early 1980's, software engineers began to realize that there was a major problem. The artwork required to create data flow diagrams, entity-relationship diagrams, structure charts, state-transition diagrams, and other graphic models was overwhelming. In addition to the work required to create and maintain the diagrams, classical structured analysis requires a great deal of work to verify the diagrams to ensure that they are complete and consistent. This verification had to be done manually and because it was labor intensive and boring, it tended to be error prone. Consequently, many of the specification errors that should have been found were not.

Many of these problems can be solved with proper automated support; this was well-known when structured analysis was first introduced, but the cost of automation was far higher than most organizations could afford. However, the development of powerful graphics workstations in the mid-1980's led to a whole new industry known as CASE (Computer-Aided Software Engineering).

It is estimated that about 25 to 30% of software engineers have access to personal CASE tools. Many designers, especially those working in real-time and embedded systems, remain skeptical that CASE tools will help them develop better systems. This skepticism can be traced to some of the first CASE tools of 20 years ago. They failed to live up to promises of easing maintenance and ensuring error-free code, making potential purchasers wary. However, it is clearly the way of the future and we can expect that all professional Software Engineers as well as those paying for the software will insist on such tools as time goes on.

The two primary objectives of CASE tools are:

1. Higher Quality Systems
    - a. Analysis - Requirements must be:
        - 1) Correct
        - 2) Consistent
        - 3) Complete
        - 4) Realistic
        - 5) Needed
        - 6) Verifiable
        - 7) Traceable
    - b. Design - The Design must be:
        - 1) A solution to the problem
        - 2) Modular and well-structured
        - 3) Portable
        - 4) Easy for implementers to understand
        - 5) Well-documented
        - 6) Cross-referenced with the requirements
    - c. Programming - the Programs must be:
        - 1) Structured
        - 2) Efficient
        - 3) Simple
        - 4) Well-documented
        - 5) Testable
        - 6) Easily modified

2. Improved Productivity - Use the power of the computer to improve the productivity of the software engineer and therefore lower the cost of software development projects.

With these objectives in mind, the rest of this paper will give an overview and introduction to CASE tools, their characteristics, uses, etc. It is then followed by an evaluation of various specific CASE tools.

CASE TOOL OVERVIEW[2]

CASE tools provide automated support for carrying out various tasks in the Software Development Life Cycle (SDLC). A CASE tool refers to any software tool that provides the software engineer with automated assistance in the creation, maintenance, or project management of software systems.

CASE toolkits combine CASE tools into an integrated set that serves to automate one particular task of the software development process, such as the design phase. Therefore, toolkits are usually referred to by the particular SDLC phase they support.

CASE workbenches or environments combine CASE tools into an integrated set that serves to automate several tasks across the SDLC. The unique feature of a workbench is the fact that the output of one life cycle phase is automatically carried over and continued into the next phase. A complete CASE workbench supports the following tasks of the software development process:

- Graphically represent design specifications
- Track and cross-reference system information
- Report on system information
- Build prototypes and system simulations
- Generate program code
- Generate programming documentation
- Enforce adherence to standards

CASE Methodology Companions are a special set of integrated CASE tools which serves to automate the software development process according to the rules of a specific structured methodology. Methodology companions can be either a toolkit or a workbench. The tools require the software engineer to adhere to a predetermined series of development steps. Functions within the methodology companion will not allow the developer to skip over a necessary step. In addition, the methodology companion checks the output of a process and confirms that it meets the required specifications and standards.

Some CASE toolkits are designed to work under the direction of a specified structured methodology. For example, the Analyst/Designer toolkit from Yourdon, Inc. only supports the Yourdon Structured Design methodology, while most recently developed CASE tools support multiple methodologies such as System Architect from Popkin Software & Systems, Inc. Some of the most popular methodologies are:

* **Yourdon/DeMarco** - Structured methodology

* **Gane/Sarson** - Structured methodology

* **Ward/Mellor** - Real time extensions to the Structured methodology

* **Rumbaugh[3]** - Object oriented methodology using the Object Modeling Technique (OMT)

* **Shlaer/Mellor** - Object oriented methodology

• **Information Engineering**[4] - Methodology introduced by Clive Finklestein and later popularized by James Martin. The methodology addresses the lack of cohesiveness between the different phases of other SDLC's. It addresses strategic planning, as well as analysis and design, and is data driven. This focus allows the developer to concentrate the model building activity on the more stable facet of an organization, its data. The key to the power of Information Engineering is its degree of integration. While many of the techniques used in Information Engineering originated elsewhere, Information Engineering clearly defines how deliverables from one technique relate to deliverables from other techniques within and across development stages.

• **SSADM**[5] - Structured Systems Analysis and Design Methodology is an enhanced rewrite of the Learmonth and Burchett Management Systems System Development Methodology (LBMS-SDM) which is a highly structured, data driven method of systems analysis and design that covers all stages of system development. The input to the methodology is an initial study report and the outputs are stated as program specifications, user procedures, operating instructions and the file design or data base schema. It incorporates highly defined procedures for both logical and physical design, and relies heavily on the data analysis approach. The major techniques used in the methodology are data flow diagrams, logical data structuring technique, third normal form data analysis, a data dictionary, walkthroughs, and reviews. These techniques are based on the work of DeMarco, Yourdon, Gane and Sarson, Martin, Codd, and Bachman.

• **DOD-STD-2167A**[6] - Defense System Software Development, establishes a standard for planning and controlling software development on large software development projects. The standard is primarily used within the United States for engineering-oriented system acquirers (e.g., NASA, Department of Transportation, Department of Defense) and contractors. Some countries outside the US use the standard too, generally for work under contract with a US firm developing a system in that country.

US industry and Government personnel worked for several years to establish a uniform standard for large software development projects. The US Government released an initial version of the standard, DOD-STD-2167, on June 4, 1985 and then on February 29, 1988 a revision, DOD-STD-2167A, was released. The revision resulted in a publication that was shorter and simpler. The earlier 2167 standard presented the classical "waterfall" model. Whereas 2167A is development cycle independent and software method independent. The contractor defines how to organize activities over time, and includes the description in the Software Development Plan. This means that the contractor can select his process model -- waterfall, rapid prototype, multiple build, spiral, or some hybrid of these. 2167A allows the contractor to choose his software method, whether it be object oriented or structured or some in-house version, and it too is documented in the Software Development Plan.

Integrated CASE Environment is a suite of linked tools that addresses different aspects of the development process. Typically, these tools are produced by different vendors. Tool integration is intended to produce complete environments that support the entire software development process. There are four types of integration:

1. Presentation integration --sharing a look and feel from the user's perspective.

2. Data integration --sharing data among tools and managing the relationships among data objects produced by different tools.

3. Control integration --event notification between tools and the ability to activate tools under a program's control.

4. Platform integration --inter-operability via distributed processing which make it possible to use network-based file systems and network communication to convert and transmit files from one execution environment to another.

Software development tools are classified as "**vertical**" - tools that are used for a specific phase of the process- and "**horizontal**" - tools that are used throughout the software development process. In the following figure the various tools are depicted in an overall framework that shows a model of an integrated tool environment:

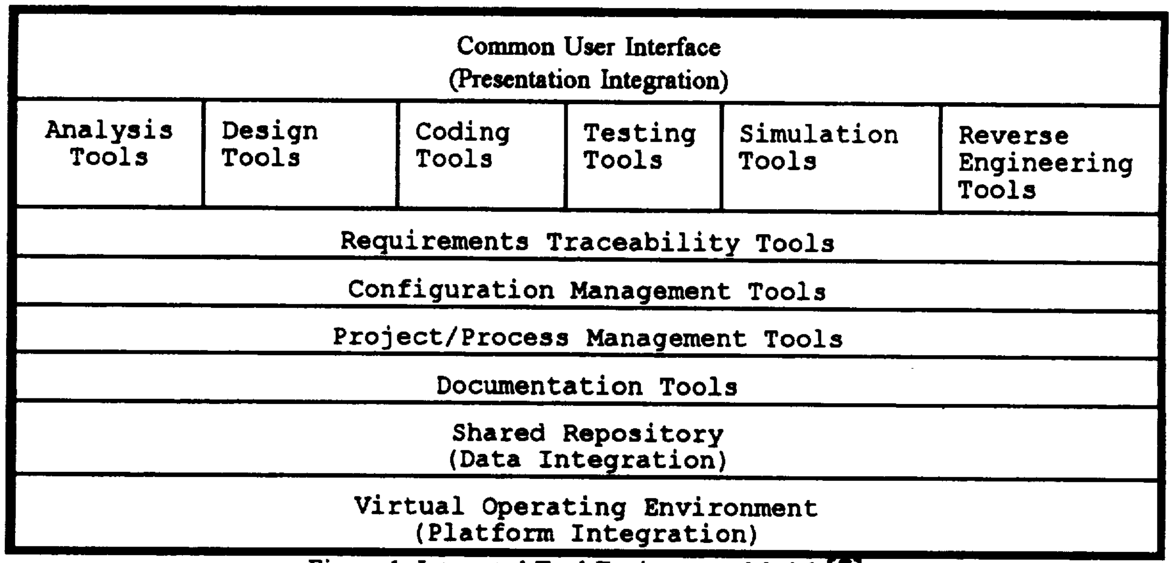
Common User Interface
(Presentation Integration)
Analysis Tools
Design Tools
Coding Tools
Testing Tools
Simulation Tools
Reverse Engineering Tools
Requirements Traceability Tools
Configuration Management Tools
Project/Process Management Tools
Documentation Tools
Shared Repository
(Data Integration)
Virtual Operating Environment
(Platform Integration)

Figure 1. Integrated Tool Environment Model [7]

Vertical CASE tools assist software engineers by automating the tasks specific to a phase of the software development life cycle. Vertical CASE tools support the automatic creation of the following:

* **Data Flow Diagrams**(DFD). Software engineers use data flow diagrams for structured systems analysis and object oriented analysis. Data flow diagrams allow a user to model the system from a "functional" perspective. They do this by first defining the system at the top level to show its overall purpose and how it interacts with external objects. Then, the system is divided into individual components or processes using decomposition techniques. Every process transforms the incoming data, then passes its output data to another process or to a data store. For primitive processes that are not decomposed, the user generates a process specification ("mini-spec") from a user-modifiable template. In integrated workbenches, this information is stored in a central repository and lays the groundwork for design and implementation. The purpose of the Data Flow Diagram is not to specify, but to declare. It declares component processes that make up the whole, and it declares interfaces among the components. Although the DFD is the primary emphasis in structured analysis it has less emphasis in object oriented analysis.

* **Hierarchical Data Structure Diagrams (HDSD)**. Software engineers use hierarchical data structure diagrams during the design phase to decompose data flows and data stores that are defined in data flow diagrams and structure charts.

* **Entity Relationship Diagrams**(ERD). Entity relationship diagrams were added to the structured analysis method by Yourdon to model the system's stored data layout at a high level of abstraction. Yourdon called this modification, "modern" structured analysis. It highlights relationships between data stores on the DFD that would otherwise be seen only in the process specification. ERDs are also used during the design phase to model system data by defining entities and their relationships, particularly when the application is data-intensive.

* **Control Flow Diagrams**. Control flow diagrams are used to model the event-driven aspects of real-time systems. Control flow diagrams show the interaction between discrete-valued control signals and processes. They describe the variety of pathways that the program can take depending on external events.

* **Finite or State Transition Diagrams**. Finite or state transition diagrams are used to model the behavior of event-driven systems. State transition diagrams allow users to view the system from an "event-control" perspective. They show the conditions and actions that cause a system to change from one state to another.

* **Structure Charts**. System designers use structure charts for structured design. Structure charts allow users to define the software architecture, including the interfaces between modules. Modules are defined hierarchically by specifying which modules "call" other modules.

* **Object Diagrams**. The Object Diagram is typically the first diagram built using the Object Modeling Technique of the Rumbaugh Object Oriented Analysis methodology. This diagram contains analysis and design information about how objects in the system are related to one another. It is a graph whose nodes are object classes and whose arcs are relationships among classes.

Horizontal CASE tools used throughout the software development life cycle are described as follows:

* **Shared Repository or Data Dictionary.** The data dictionary (also called Encyclopedia by KnowledgeWare's Application Development Workbench) is a repository that contains entries for all data elements, objects, etc. that are pertinent to the system, with precise, rigorous definitions so both the user and analyst will have a common understanding of all inputs, outputs, components of stores, and intermediate calculations. The Data Dictionary is used by CASE tools for centralized storage of intelligent information about the system. Information about the system is collected in this repository throughout the development life cycle allowing ease of transition among the planning, analysis, design and construction phases of the life cycle. This repository is also constructed with reverse engineering tools allowing the automatic creation of tools to assist with system understanding and forward engineering.

* **Requirements Traceability.** Requirements Traceability tools produce reports to enable software developers to track original customer requirements throughout the early phases of the software development process. Tracing customer requirements enhances management control and improves communication between software developers and their customers.

Examples of Requirements Traceability reports include:

- ☐ objects satisfying a particular requirement
- ☐ structure chart modules traced back to data flow processes
- ☐ data flow processes linked to structure chart modules

* **Reverse Engineering.** Although shown in the above chart as a vertical tool because it is used after code is produced, reverse engineering tools can be used to support the entire life cycle. It is this view that the Cadre tool Ensemble has. Perhaps the title of Software Rejuvenation would be more fitting. Software Rejuvenation addresses the challenge of software system maintenance by trying to increase the overall quality of an existing system. It looks back at the workproducts of a system to try to derive additional information or to reformat them in a more understandable way. There are several aspects of software rejuvenation to consider, including:

■ Redocumentation which is the static analysis of source code whose product is additional information that assists maintainers in understanding and referencing the code. The analysis does nothing to transform the actual code; it merely derives information.

■ Restructuring which actually changes the code by transforming ill-structured code into well-structured code.

■ Reverse engineering which looks back from the source code to the products that preceded it, re-creating design and specification information from the code.

■ Re-engineering which is broader still, is where the existing system is reverse engineered and then "forward engineered" to make changes to the specification and design to complete the logical model; a revised system is then generated from the new specification and design.

Interestingly, NASA is very interested in this area of CASE tools not necessarily for the maintenance of systems but for the evaluation of systems produced by contractors. Areas where these tools could be helpful to NASA would be:

■ Requirements traceability. The tools could be used to reverse engineer the delivered code back to the design specifications. The tool produced specifications could be compared to the original design specifications and differences explained, justified, or corrected. Using special requirements traceability tools which span the entire life cycle, design specifications can be traced back to the requirements, given that the CASE tool was used to produce the original requirements.

■ Quality Assurance. The tools could be used to analyze the delivered code, generate comprehensive test cases, validate that all important paths were tested, generate system metrics, etc.

Also, interestingly, these tools can be used by the contractors to generate the required documentation. It is possible to reverse engineer a system of source code back to design specifications to include items such as structure charts, control flow diagrams, cross-references, library calls, etc. Then using special documentation tools with DOD-STD-2167A templates generate the required design documentation. If the CASE tool was used throughout the project's life cycle, requirements through code generation, CASE documentation tools are able to extract, assemble and format total system documentation . Some CASE vendors see this as a software engineer productivity tool. "No more having to do that awful program documentation." Let the tool do it.

CASE TOOL SELECTION CRITERIA

Over the past decade a broad array of software tools has been introduced to the software engineering marketplace. Software tools that support analysis, design, testing, maintenance activities, configuration management, and even quality assurance combine to create a CASE environment. Unfortunately, vendor claims for specific tools and the performance of these tools in actual practice are often dramatically different. For this reason, it is very important to assess tools carefully using the criteria in [8] as a guide.

A Case Tool Checklist

The above referenced generic tools checklist can be used effectively when CASE is considered. However, CASE systems have a set of specific attributes that should be considered as well. A specialized checklist for CASE selection can be found in [9]. Probably no existing CASE system will exhibit all of the attributes listed in the above reference. The best systems offer a reasonable subset. Use the attributes checklist as a guide in evaluating vendor offerings.

A COMPARATIVE ANALYSIS AND EXPERIENCES WITH SOME SPECIFIC CASE TOOLS

As a Summer Faculty Fellow working for NASA, the author had the opportunity to do research in the CASE tools area. The research involved reading the many texts that there never seem to be enough time for back at the university, discussing current CASE issues with fellow Faculty Fellows, attending CASE product seminars and demonstrations and getting some actual hands-on experimentation with real CASE tools. The author is grateful to the companies that loaned evaluation copies for this research. The following is a report on the results of experimenting with each tool.

Software Through Pictures (StP), C Development Environment from Interactive Development Environment is written for the UNIX environment and was installed on a Sun Sparc 2 running in a Sun OS 4.1.2 environment.

StP is an integrated, open environment, built upon a multi-user, object-based repository that facilitates the integration of StP tools with other development tools in a consistent, seamless manner. StP provides open and published interfaces including diagram file formats and repository schema, as well as user-modifiable configuration and message files. The StP product includes a family of graphical editors that implement several commonly used analysis and design methods, as well as code generators. For example, there are analysis tools that support the structured analysis using either the notation developed by Yourdon/DeMarco and the notation developed by Gane/Sarson. It also supports modeling of real-time systems. StP products provide user-modifiable templates for object annotation, documentation preparation, code generation, SQL schema generation, and program design language.

Paradigm Plus, from Protosoft Inc. , is a configurable CASE tool that uses an object-oriented information model to provide support to a wide range of software engineering activities practiced throughout the application development life cycle.[10] Paradigm Plus comes with a set of application development rules and procedures including:

- Rumbaugh OMT - Object Modeling Technique
- Rumbaugh 93 - latest extensions to OMT
- Booch OOD - Object Oriented Design
- HP Fusion - Object Oriented Analysis and Design Method
- ProtoSoft OOAD - Object Oriented Analysis/Design
- EVB - Ada Object Oriented Design
- others

The set of application development rules and procedures are implemented by using the Paradigm Plus

CASE Development Kit (CDK) technology. The CDK *Paradigm Definition Language* (PDL) provides the ability to specify a particular set of rules and procedures to include:

- Meta Model
- Rules and Semantics
- Diagram Notations
- Menus
- Toolboxes
- Icons
- Dialog Boxes
- Hypertext Help
- Standard Reports
- Code Generation

Paradigm Plus incorporates many features that set it apart from conventional CASE tools. It is Method Independent, allowing a software engineer to choose the method that is most suited to a given project. It is language independent, allowing the software engineer to specify a neutral definition of the software design, and have it automatically mapped to language specific constructs at code generation time. Paradigm Plus is platform independent, being available on Windows, Solaris 1.x and 2.x, HP PA-RISC and IBM RS/6000. Paradigm Plus stores all information about a project in an integrated, active Object Repository.

Teamwork, Ensemble [11], from Cadre Technologies, Inc. is written for both the OS/2 and the UNIX environment. The Teamwork product set is comprised of a number of integrated development tools that support the full software development life cycle. All tools support multi-user, multi-windowed, multi-tasking environments. All Teamwork tools share a common repository for each project, known as the Teamwork Project Database, which serves as a repository for data dictionaries, diagrams, process specifications, and project management data. The Teamwork series includes Teamwork/IM, Teamwork/SA, and Teamwork/RT, systems analysis tools which support information modeling, systems analysis, and real-time analysis, respectively. Other Teamwork analysis/design tools include Teamwork/SD, which supports structured design, Teamwork/Ada, a design, navigation and documentation tool for Ada projects, and Teamwork/OOD, which provides direct support for the object-oriented design (Schlaer/Mellor).

Teamwork supports code generation via their Ada and C Source Code builders. Tool sharing and portability is provided by Teamwork/IPSE which enables all Teamwork products to integrate with other software applications. A new member of the Teamwork family is Teamwork/SIM, which integrates dynamic modeling and real-time simulation into the Teamwork environment, providing verification of real-time system behavior.

Ensemble is a modular tool suite that automates software development and maintenance for C professionals. Integrating a wide range of functions required for understanding, constructing, testing, and documenting code under a graphical user interface and shared database, Ensemble complements existing tools by fitting easily into current development environments and enabling a phased adoption of its modules. For a detailed product description, see CADRE's teamwork Product Overview[12].

EVALUATIONS/RECOMMENDATIONS

1. Product Comparisons and Comments. Because of limited time and hardware availability, the author was not able to explore in-depth the Software through Pictures (StP) and the Paradigm Plus products. With that in mind, the following are some general comments about the products.

The **StP** product was an Upper CASE tool. I experimented with the graphical editors normally used in the Requirements phase to draw Data Flow Diagrams, Entity Relationship Diagrams, etc. I found them a bit cumbersome to use. You definitely need a strong Unix System Administrator to install the package. IDE has recently announced a Reverse Engineering tool and wanted to make it available for our evaluation. However, time and IDE policy prohibited this. IDE product documentation was excellent. The marketing literature is also excellent. IDE is selling I-CASE, but we only know of an Upper CASE tool and the recently announced Reverse Engineering tool.

The **Paradigm Plus** product supports OOA (Object Oriented Analysis) using the OMT (Rumbaugh) methodology, both of which are new and seem very complex to the author. The product installed without any problems. The version we tried runs in a PCDOS/Windows environment. We had draft documentation for a new up coming release. We found the product easy to use with the only negative aspect being that charts printed on the printer were low quality. The lines and text were too thick. We were unable to tell if this could be controlled by some parameters. Although the sales literature lead you to believe that Paradigm Plus will generate code, no indication of this function could be found in our evaluation copy literature.

CADRE's TEAMWORK/ENSEMBLE was the product that most time was spent examining. Most of the time was spent with the reverse engineering tool ENSEMBLE. We found that the complete package was indeed an I-CASE tool. A CADRE marketing representative installed and demonstrated the product. The only trouble we had was installing a new key file. This is a file used by the system to control the licensing of the products. Checksums are generated from the information in the key file and compared by predefined algorithm. Therefore, the data in the keyfile has to be "exactly" correct. The first time, after many hours with CADRE's help-desk (hot-line), they had to email us a file. We were more successful the second time when CADRE gave us an extension on our evaluation time.

We found their people on the hot-line to be outstandingly helpful. They were courteous, prompt, and guided us through several problems. They also followed up to ensure we had indeed solved our problem.

We reversed engineered two programs with very satisfactory results. One was the huge C program, Dynamic Environment Communications Analysis Testbed (DECAT), the other a modest C program written by another faculty fellow. We encountered no problems and produced some useful documentation from the reverse engineered repository.

Another faculty fellow experimented with the Upper CASE tool to design and generate an Ada program. This effort is still underway and will be completed after the author leaves.

2. Recommendations

My recommendations for the Systems Design Section of the System Engineering Branch of the Tracking and Communication Division at NASA JSC regarding CASE tools fall into two categories, internally developed and maintained systems and externally developed and maintained systems.

a. Internally developed and maintained systems. First of all I recommend you appoint someone to be your software engineering expert. Then, you must adopt a methodology. The gurus agree unanimously that a methodology is an absolute requirement for software success. Ed Yourdon says "It is better to pick any methodology than it is to pick mine."[13] Gerald Weinberg states "Bringing in a CASE tool where there is no methodology, where anybody can build systems any way they want, is guaranteed to be a waste of time." He goes on to say "A methodology is a way of thinking, not a substitute for it." The learning curve for methodologies is typically from 13 to 24 months.[14] Because you will need some experience and knowledge of methodologies and CASE tools to manage the externally developed and maintained systems, this will prepare you for managing the external projects better. Although the author is not totally up to speed on OOA and OOD, from the research done so far indications are that the organization should be adopting one of the Object Oriented Methodologies. As Object Oriented Programming brings promises of improved programming concepts such as code reusability, OOA promises similar benefits that are described in the following reference.[15]

After choosing a particular methodology, then a CASE tool should be purchased. The tool should be an I-CASE tool that includes reverse engineering. I recommend that CADRE's Teamwork/Ensemble package be seriously considered. An approximate license cost is $70,000 for a single user plus an annual maintenance. Official price quotes should be obtained directly from CADRE. CASE tools, especially I-CASE are large complex systems. They are not intuitive for a simple reason. They automate complicated design techniques. (The most advanced word processing software will not make you a good author.) Therefore, I also recommend that your software engineering expert obtain detail knowledgable on CASE tools by attended formal training for both the methodology and the CASE tool. Using the tool/methodology will prepare your organization for dealing with the following:

b. Externally developed and maintained systems. The real issue with recommendations in this area is the DOD-STD 2167A, Defense System Software Development standard currently used by NASA. According to Coad/Yourdon[16] 2167A is model and method free. The contractor is allowed to select his process model - waterfall, rapid prototype, etc, and also select his method - structured, object, etc. However, this selection must be agreed upon by the Government. Therefore, it would seem that NASA could specify the model/method and CASE tools to be used by the contractor. The author recommends that NASA use the waterfall model and object oriented methodology. For really new technologies a variation of the waterfall model could be used where prototyping was used to gather requirements. For project management reasons, the author is opposed to the pure prototyping models. For a detailed mapping of OOA to DOD-STD-2167A, see the above reference.

Reverse engineering tools seem to be model/method independent, and only source program language independent. Since most programs written for NASA is written in either Ada or C, a reverse engineering tool could be used by NASA to gather metrics, verify "hot" spots for testing and even generate acceptance test cases. Again, CADRE's Ensemble tool would be very useful for these purposes.

If NASA could specify to the contractor the model/method/I-CASE then additional benefits could be gained such as the presumed increased contractor software engineering productivity, requirements traceability, software and analysis reuse, increased system reliability, etc. It is strongly recommended that NASA set this specification as a near term goal.

REFERENCES

[1] Van Tyle, Sherrie. Electronic Design, May 13, 1993.

[2] SRA: Using CASE Methodologies. Science Research Associates, 1991.

[3] Rumbaugh, James, Object-Oriented Modeling and Design, Prentice-Hall, 1991.

[4] Madison, R. N., Information System Methodologies, Wiley Hayden, 1983.

[5] ibid.

[6] Coad/Yourdon, Object-Oriented Analysis, second edition, 1991

[7] IDE Product Catalog 90-00713-492-30M, Interactive Development Environments. IDE, 1992.

[8] Pressman, R. S., Making Software Engineering Happen. Prentice-Hall, 1988.

[9] ibid.

[10] Paradigm Plus User's Guide, Revision 1.1, June 1993, first edition.

[11] CASE Trends, Volume 5, Number 4, Summer 1993.

[12] CADRE's teamwork Environment Reference Manual, Release 4.0, 1990.

[13] CASE Trends, Volume 5, Number 5, July 1993, pp56.

[14] ibid

[15] Coad/Yourdon, Object-Oriented Analysis, second edition, 1991

[16] ibid, Appendix B